Apple II
Technical Notes

Developer Technical Support

Apple IIGS
#84: TaskMaster Madness

Written by: C.K. Haun <TR> July 1990

This Technical Note discusses the enhancements made to `TaskMaster` in System Software 5.0.

`TaskMaster` has been expanded to handle extended control actions and give you more information about events in System Software 5.0. This Note discusses some features of the expanded `TaskMaster` and `TaskMasterDA`, and how you can best exploit the new features in your applications.

Stop Making It So Difficult

Developers just want to work too hard. You get a neat new thing like the expanded `TaskMaster`, and you still want to do all the work yourself. The new `TaskMaster` does nearly **everything** for you, as long as you treat it correctly.

What this means is you do **not** have to call `FindControl`, `TrackControl`, `TEIdle`, `LEKey`, handle keystrokes for controls, keep track of click counts, or any of the other mundane event management tasks unless you specifically want to perform actions that `TaskMaster` does not perform. For the standard controls and situations this means that you do not have to do **anything**.

The magic keys to this life of freedom and ease are the five newly defined `taskMask` flag bits, labeled in the interfaces as `tmContentControls`, `tmControlKey`, `tmControlMenu`, `tmMultiClick` and `tmIdleEvents.` This Note looks at what the new bits do for you, but first a word of warning.

Warning: If you set **any** of these new bits, `TaskMaster` assumes you are using the new extended task record. This means that you cannot just go into an older program and set these bits and expect your program to work successfully. You also **must** allocate the additional space for the extended portion of the task record. If you do not, `TaskMaster` puts task data in areas that you do not expect, and Bad Things happen.

Bits 'o This, Bits 'o That

Click Bits

`tmMultiClick` tells `TaskMaster` to keep the new "click information" fields in the extended task record updated. This allows you to have `TaskMaster` keep track of multiclick events; the `wmClickCount` field is one, two or three depending on whether the last action was a single, double, or triple click. In fact, if you can click your mouse button fast enough, you can time quadruple clicks, sextuple clicks, or as high as you want, although anything over triple-clicking is nearly impossible for users to consistently manage. `wmClickCount` just gets incremented by one when the click falls within the double time interval. `wmLastClickTick` is updated with the system tick value at last click. `wmLastClickPt` contains the location of the last mouse click. `TaskMaster` calls `GetDblTime` internally to determine the correct time intervals for these values.

Idle Bits

`tmIdleEvents` tells `TaskMaster` to call the idle routines for controls that need idle events, like TextEdit controls and LineEdit controls. This also means that only the active control is blinking a cursor, since `TaskMaster` is working with the target bits of the extended control records to keep track of which TextEdit or LineEdit control is active and switching the target control in response to mouse clicks and Tab keypresses. This is also the area where you tell `TaskMaster` how to highlight your window controls. Using the Control Manager calls `MakeNextCtlTarget` and `MakeThisCtlTarget` allows you to specify which LineEdit or TextEdit control is active. You can use these calls to highlight input errors the user has made. For example, if someone has entered text in a LineEdit control that requires a number, you can alert the user if he enters non-numeric characters with an `Alert` or `AlertWindow` call. You can then direct the user to the LineEdit control that contains the bad entry by calling `MakeThisCtlTarget` with the handle of that LineEdit control. This deactivates any other target control and moves the insertion point to the LineEdit control that needs the correction.

Contentious Bits

`tmContentControls`, `tmControlMenu` and `tmControlKey` bits are the real workhorses of the expanded `TaskMaster`.

When the `tmContentControls` and `tmControlMenu` bits are set, `TaskMaster` handles the mouse activity side of events—tracking, highlighting or popping-up the selected control. If the control is a radio button, check box, pop-up menu or list control, `TaskMaster` also performs the correct action for the click, either setting the control value, scrolling the list, setting the pop-up menu to the selected item, and so on. `TaskMaster` then returns a `taskCode` of `wInControl` ($21). The control handle is stored in `wmTaskData2`, the part code of the part selected in `wmTaskData3` and the control ID is in `wmTaskData4`. For many of the controls in your windows your application needs to take no further actions, `TaskMaster` has set the control values. When the user closes the window or clicks on a button that causes an action, you can then read the values of all the controls you care about at that point and do what you need to do, instead of keeping track as the user manipulates controls.

The last new bit, `tmControlKey`, works with the `tmControlMenu` bit to handle key events for your extended controls.

When a key event occurs, `TaskMaster` sends the event to the internal routine `TaskMasterKey`. `TaskMasterKey` first looks at the `tmMenuKey` bit (which has been in `TaskMaster` since the Window Manager was implemented). If it is set, then `TaskMaster` tries to handle the event as a menu event, calling `MenuKey` for the current menu bar.

Note: This also means that any key equivalents in your main menu bar (across the top of the desktop) take precedence over key equivalents in your window controls.

If this fails (or that bit is not set) and `tmControlKey` is set, then `TaskMasterKey` polls the controls in the currently open window for any controls that would like this keystroke, either for controls with a `keyEquivalent` field or a pop-up menu control with key equivalents for menu items. If it finds a control that wants the key event, it is handled very much like a mouse event. The action for the control is performed (checking a check box, for example) and the `wmTaskData` fields are filled as they would be for a mouse click, and an event code of `wInControl` ($21) is returned. If a key event did occur, you can differentiate it from a mouse event by looking at the `wmWhat` field of the `taskRecord`. Even though a `wInControl` event code was passed back by `TaskMaster`, the `wmWhat` field is either $0001 or $0003, the former for a mouse down event and the latter if a keystroke stimulated the `wInControl` event.

Even More Bits

All these new features rely very heavily on the changes made to the Control Manager in System Software 5.0. Many of the `TaskMaster` features, keystrokes, target controls, and so on **only** work if you have the `moreFlags` bits set correctly in your control definitions. If you are having difficulty with new `TaskMaster` features, check your control definitions against the information in the Control Manager chapter of Volume 3 of the Apple *IIGS Toolbox Reference* and Apple IIGS Technical Note #81, Extended Control Ecstasy.

Don't Get Goofy

There are some dangers in these new features, of course. By allowing built-in key equivalencies for almost all the controls that can exist in a window, it may be tempting to define key equivalents for everything, and create weird and unusual key combinations for your controls. **Please** remember the *Human Interface Guidelines* (specifically Human Interface Note #8, Keyboard Equivalents) and keep your use of keystroke equivalents to a minimum. Multimodifier keystrokes (Command-Option-Shift, for example) do not enhance the user's experience and can be very confusing.

NDAs Can Have Fun Too

`TaskMasterDA` has also been added to the Window Manager, providing your new desk accessories (NDAs) with the same kind of `TaskMaster` support your applications have. This lets you easily use extended controls inside NDAs, following the same basic rules as in an application. There are only a few things to worry about.

What Does That Stack Picture *Really* Mean?

The input to `TaskMasterDA`, as shown in Volume 3 of the Apple *IIGS Toolbox Reference*, is as follows:

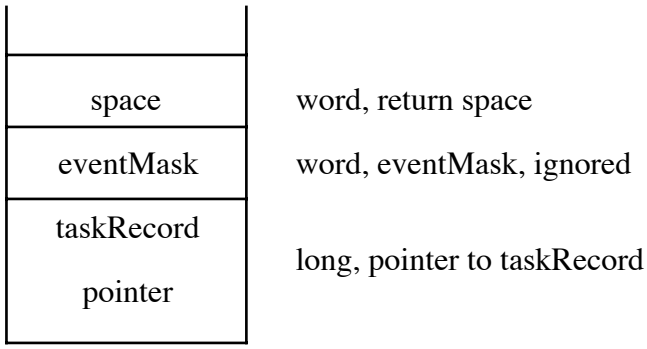

Figure 1–TaskMasterDA Stack Picture

The call returns a word value, the `taskCode`. The space and `eventMask` are self-explanatory. The book tells you that the `eventMask` is ignored, which makes sense since the host application has already gotten the event and you have already specified an `eventMask` in your NDA header, so you can use any value here. The `taskRecPointer` causes the confusion.

You do not pass a blank event record. When your NDA's action routine is called, the `Y` and `X` registers contain a pointer to the current event record with which the NDA is working. `TaskMasterDA` is filling that `taskRecord` with some information, so you want to move it into your NDA's data area so you can work with it later:

```
phy
phx                     ; push the pointer that was passed to us
pushlong  #NDArecord    ; the space in my NDA for the extended event record
pea       0
pea       16            ; only 16 bytes, the original taskRecord size
_BlockMove
```

It is **very** important that you only move 16 bytes. `TaskMasterDA` can act erratically if the extended portion of the event record has been filled with nonsense values. This can happen if your NDA is running in an application that does not use the extended task record and you are copying non-task data into the extended portion of the task record. By the way, as you are debugging your NDA and you run into situations where the `wmTaskData` field values are weird, this is more than likely the problem.

Also remember to make sure the `wmTaskMask` field in your NDA's `TaskRecord` is set and the extended portions of the `TaskRecord` are zeroed out before your NDA starts running; you want to set all these fields in your NDA's `INIT` routine.

Now you can call `TaskMasterDA`:

```
pea       0             ; return space
pea       $FFFF         ; eventMask, ignored
```

```
pushlong  #nDArecord    ; our NDA event record
_TaskMasterDA
pla                     ; event code returned
```

You can then process the event in a convenient way. Remember again that `TaskMaster` has already done the control tracking for the controls in your NDA window. You have the same multiclick information, control handles and IDs.

Conclusion

`TaskMaster` is a wonderful thing that makes any programmer's job easier. So let it work **for** you. Learn the capabilities of the new fields and new controls, and take advantage of them. Let `TaskMaster` take care of the system details, while you concentrate on the features that make your application special.

Further Reference

- *Apple IIGS Toolbox Reference*, Volumes 1 through 3
- Apple IIGS Technical Note #81, Extended Control Ecstasy
- Human Interface Note #8, Keyboard Equivalents